Convenient data manipulation with dplyr

Introduction

dplyr (https://github.com/hadley/dplyr) is a data manipulation package for the R (http://www.r-project.org) language. Although it was first released this year (2014), it has already become an essential tool for many. In fact, for actuaries learning R, I think it's worth picking up right away and considering a core part of the R language. Three reasons for this are:

1. dplyr's syntax is more intuitive than the basic R data manipulation functions.
2. Actuarial tasks frequently involve simple manipulations of tabular data, and dplyr excels at this.
3. dplyr is fast and scalable. You won't need to switch to a different tool if you start dealing with bigger data sets.

This blog post will introduce dplyr using actuarial examples. We'll start with a small data set to show the basic syntax. Then we'll examine dplyr's speed on a larger data set.

There are several good introductions to dplyr, including the basic vignette included in the package (http://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html). We won't repeat most of that here, and will stick to illustrating dplyr on a sample actuarial data set. For details of the various dplyr functions, please refer to that vignette or the reference manual (http://cran.r-project.org/web/packages/dplyr/dplyr.pdf).

Small toy example

Suppose we have a data frame called `quarter.df` that contains aggregate premium and loss by quarter for two years:

```
set.seed(0)
quarter.df <- data.frame(year=c(rep(2011, 4), rep(2012, 4)),
                         quarter=rep(1:4, 2),
                         prem=rnorm(8, mean=5e6, sd=5e6 * .05),
                         loss=rnorm(8, mean=5e6 * .7, sd=5e6 * .7 * .3))
```

year	quarter	prem	loss
2011	1	5315739	3493944
2011	2	4918442	6024886
2011	3	5332450	4301773
2011	4	5318107	2661040
2012	1	5103660	2294960
2012	2	4615012	3196065
2012	3	4767858	3185824
2012	4	4926320	3067914

As an actuary, you may wonder: What are the loss ratios by quarter and by year? Can we compare the yearly and quarterly loss ratios?

dplyr can help us do this. The dplyr elements we'll use here are:

1. group_by - group a table according to a column
2. summarize - distill a table that has already been group_by'd
3. inner_join - merge one table to another, like SQL's inner join
4. %>% - pipe operator (really part of the magrittr package, but you'll get it if you load dplyr)

To get the loss ratios by year we will group the quarterly data frame into a yearly one:

```
year.df <- group_by(quarter.df, year)
year.df <- summarize(year.df, yr.prem=sum(prem), yr.loss=sum(loss))
year.df <- transform(year.df, lr=yr.loss / yr.prem)
year.df
```

```
##   year  yr.prem  yr.loss     lr
## 1 2011 20884737 16481644 0.7892
## 2 2012 19412851 11744763 0.6050
```

Note how we kept redefining year. dplyr often lends itself to chained assignments using the pipe operator %>%. Conceptually the operator is very simple; it just lets us write "x %>% f(…)" instead of "f(x, …)". It doesn't really do anything, it just lets us write functions in a different syntactic order.

But sometimes that makes a huge difference! The following two blocks of code are both equivalent to the previous one. Which would you rather read?

```
year.df <- transform(summarize(group_by(quarter.df, year),
                               yr.prem=sum(prem), yr.loss=sum(loss)),
                     yr.lr=yr.loss / yr.prem)
```

```
year.df <- (quarter.df
            %>% group_by(year)
            %>% summarize(yr.prem=sum(prem), yr.loss=sum(loss))
            %>% transform(yr.lr=yr.loss / yr.prem))
```

Finally, we can combine our yearly and quarterly data frames to compare loss ratio by quarter.

```
quarter.df <- (quarter.df
               %>% transform(lr=loss / prem)
               %>% inner_join(year.df[, c("year", "yr.lr")], by="year")
               %>% transform(diff.from.year=lr - yr.lr))
```

year	quarter	prem	loss	lr	yr.lr	diff.from.year
2011	1	5315739	3493944	0.6573	0.7892	-0.1319
2011	2	4918442	6024886	1.2250	0.7892	0.4358
2011	3	5332450	4301773	0.8067	0.7892	0.0175
2011	4	5318107	2661040	0.5004	0.7892	-0.2888
2012	1	5103660	2294960	0.4497	0.6050	-0.1553
2012	2	4615012	3196065	0.6925	0.6050	0.0875

year	quarter	prem	loss	lr	yr.lr	diff.from.year
2012	3	4767858	3185824	0.6682	0.6050	0.0632
2012	4	4926320	3067914	0.6228	0.6050	0.0178

Above, the diff.from.year column is the difference between the quarterly loss ratio and the loss ratio for that year.

Larger example and speed comparisons

dplyr is relatively fast. Let's expand the example above to have 10 million rows:

```
library(tweedie)
library(lubridate)
set.seed(0)
num.rows <- 1e7
date.range <- as.Date("2003-01-01"):as.Date("2012-12-31")
day.numbers <- sample(date.range, num.rows, replace=TRUE)
policy.df <- data.frame(policy.num=seq(length.out=num.rows),
                        eff.date=as.Date(day.numbers, origin=origin),
                        prem=rnorm(num.rows, mean=100, sd=10),
                        loss=rtweedie(num.rows, mu=70, phi=100, power=1.5))
policy.df$acc.year <- year(policy.df$eff.date)
head(policy.df)
```

This time suppose we just want a listing of the 10 accident years and the total premium and loss in each year. We will test five different ways in R of computing this answer, and compare the speeds for each.

dplyr

```
system.time(
  year.big.df <- (group_by(policy.df, acc.year)
                  %>% summarize(yr.prem=sum(prem), yr.loss=sum(loss)))
)
```

aggregate

aggregate is an older R function for grouping variables in a data frame. It is included in R's core stats package.

```
system.time(
year.big.df <- aggregate(policy.df[, c("loss", "prem")],
                         by=list(policy.df$acc.year),
                         sum)
)
```

data tables

The package data.table (http://cran.r-project.org/web/packages/data.table/index.html) provides data manipulation features similar to dplyr.

```
system.time({
  policy.dt <- data.table(policy.df)
  year.big.df <- policy.dt[, list(yr.prem=sum(prem), yr.loss=sum(loss)),
                           by=acc.year]
})
```

sqldf

sqldf (https://code.google.com/p/sqldf/) is a neat package written by Gabor Grothenieck. It allows R users to write full SQL queries on R data frames. It can even be used to conveniently read/write data frames from/to a database on disk.

```
system.time({
  year.big.df <- sqldf("
select acc_year, sum(prem) as [yr.prem], sum(loss) as [yr.loss]
from [policy2.df]
group by acc_year")
})
```

ddply

ddply is from the plyr (http://plyr.had.co.nz/) and is the predecessor of dplyr. Although dplyr has replaced it for most tasks, it can still be useful for some jobs.

```
system.time(
  year.big.df <- ddply(policy.df, .(acc.year), plyr::summarise,
                       yr.prem=sum(prem), yr.loss=sum(loss))
)
```

Test results

Here is the result of the above tests (in seconds):

```
##              user  sys elapsed
## aggregate  213.88 1.15  217.42
## data.table   0.89 0.20    1.09
## ddply        3.45 1.60    5.05
## dplyr        0.77 0.00    0.76
## sqldf       48.70 6.83   55.64
```

dplyr was the fasted in this test, more than 200 times faster than the aggregate function. It's also significantly faster than sqldf and the ddply function from Hadley Wickham's previous package plyr.

However, data.table was close behind though, and would have been faster had we been using data.tables instead of data.frames to begin with. Before dplyr came out I would use data.table when I had to for speed, but I found myself always having to consult the data.table documentation because I found its syntax counterintuitive.

Conclusion

Hopefully this blog has presented a semi-realistic example of how dplyr can help process actuarial data. dplyr combines intuitive syntax, fast operation, and wide applicability into a very useful package.